1. Convert several rows into a datetime
   - df = pd.DataFrame([[12, 15, 2017, 10], [1, 15, 2018, 11]],

     columns = ['month', 'day', 'year', 'hour'])
   - df['date'] = pd.to_datetime(df)
   - df['date'] = pd.to_datetime(df[['month', 'day', 'year']])
2. # the file reading will be faster when you specify the datatype
   - df = pd.read_csv('http://bit.ly/drinksbycountry', dtype = {'continent': 'category'})
3. df = df.astype({

   'beer_servings': 'float',

   'spirit_servings': 'float'

})

4. # apply multiple aggregations on a dataframe
   - df.groupby('continent')['beer_servings'].agg(['mean', 'min', 'max']).reset_index()
5. run agg on a series
   - df['beer_servings'].agg(['mean', 'min', 'max']).reset_index()
6. run it on a dataframe
   - df.select_dtypes('number').agg(['mean', 'min', 'max', 'count']).reset_index()
7. set index
   - df.index = df['country']
   - df.loc['Angola', 'spirit_servings']
   - df.index.get_loc('Angola')
8. drop rows and columns
   - df.drop([0, 1, 2], axis = 'index')
   - df.drop('country', axis = 'columns')
   - df.drop(columns = ['country'])
9. rename
   - df.rename(columns = {'country': 'COUNTRY'})
   - df.columns = [columns names]
   - df.rename(str.upper, axis = 1)
   - df.rename(str.upper, axis = 'columns')
10. order a column
    - df = df.astype({'rate': 'category'})
    - df.rate.cat.reorder_categories(['good', 'very good', 'excellent'])
11. check for equality
    - np.nan == np.nan => False
    - np.nan.equals(np.nan) => True
12. memory usage
    - df.info(memory_usage='deep')
13. map / factorize

- titanic.Sex.map({'male': 'm', 'female': 'f'})
- titanic.Embarked.factorize()[0]
- titanic['Embarked'].factorize()[1]

14. Create dummies
    - (titanic['SibSp'] > 0).astype('int').head(10)
15. Convert to categorical
    - pd.cut(titanic.Age, bins = [0, 18, 25, 99], labels = ['child', 'young adult', 'adult']).value_counts(dropna = False)
    - pd.qcut(titanic.Age, q = 3).value_counts()
16. cross tabulation
    - pd.crosstab(titanic.Sex, titanic.Pclass)
    - pd.crosstab(titanic.Sex, titanic.Pclass, margins=True, margins_name='Total')
    - titanic.pivot_table(index = 'Sex', columns = 'Pclass', values = 'Survived', aggfunc = 'count', margins = True)
17. Group by date
    - stocks.resample('D', on = 'Date')['Close'].mean()
    - ufo.resample('Y', on = 'Time')['State'].count().tail()
    - ufo.resample('M', on = 'Time')['State'].count().tail()
18. read and save compress files
    - ufo.to_csv('ufo.csv.zip')
    - ufo.to_csv('ufo.csv.gz')
    - ufo.to_csv('ufo.csv.bz2')
    - ufo.to_csv('ufo.csv.xz')
19. interpolate missing values
    - titanic.Age.isna().sum()
    - titanic = titanic.interpolate()
20. merge
    - # check that the left keys are unique
    - pd.merge(left, right, how = 'inner', validate='one_to_many')
    - # check that the right keys are unique
    - pd.merge(left, right, how = 'inner', validate='many_to_one')
21. Transpose
    - titanic.describe().T.query('mean > 20')
22. sort a dataframe
    - df.sort_values(by = 'comments_per_view', ascending = False)[['title', 'comments', 'comments_per_view']]
23. # Extract dates from unix format
    - df['film_date'] = pd.to_datetime(df['published_date'], unit = 's')
    - df['film_date'].dt.year.value_counts().sort_index().plot()
24. convert string lists into actual lists
    - import ast

- ast.literal_eval('[1, 2, 3]')
- ast.literal_eval('[{"first": 1, "second":2}, 3]')
- df['ratings_list'] = df.ratings.apply(ast.literal_eval)

25. Vectorize a function
    - import numpy as np
    - @np.vectorize
    - def count_rating(rating_list):
    -     i = 0
    -     for dic in rating_list:
    -         i += dic['count']
    -     return i
    - df['count_rating'] = count_rating(df['ratings_list'])
    - @np.vectorize
    - def count_funny(list_of_dics):
    -     for dic in list_of_dics:
    -         if dic['name'] == 'Funny':
    -             return dic['count']
    - df['count_funny'] = count_funny(df['ratings_list'])
26. filter the dataset by the top 5 items pf a column
    - counts = df['speaker_occupation'].value_counts()
    - occups = counts[counts >= 5].index
    - df_5 = df[df.speaker_occupation.isin(occups)]
27. describe by data type
    - df.describe(include='number')
    - drinks.groupby('continent').beer_servings.describe()
28. Prefix and Suffix
    - df.add_prefix("X_")
    - df.add_suffix("_Y")
    - df.columns = df.columns.str.replace("_", " ")
29. Filter data by types
    - drinks.select_dtypes(include = ['number', 'category', 'datetime']).head()
    - drinks.select_dtypes(exclude = 'number').head()
30. Convert to numeric
    - # astype => will result of an error if there are any special character
    - pd.to_numeric(errors = 'coerce')
31. Reduce the size of a dataframe
    - # reduce the data frame size:
    - dtypes = {'continent': 'category'}
    - cols = ['beer_servings', 'continent']
    - drinks = pd.read_csv('drinks.csv', usecols = cols, dtype = dtypes)
32. create one dataframe from several dataframes

    - from glob import glob
    - booking_sheet = glob('booking*.xlsx')
    - df_total = pd.concat((pd.read_excel(file) for file in booking_sheet), ignore_index = True)
33. Duplicated
    - df_total.duplicated(keep = False).sum()
    - df_total.duplicated(keep = 'first').sum()
34. Sample fractions
    - movies.sample(frac = 0.1) # fraction
35. Filter the dataframe withour the or operator
    - # instead of filtering with the or | operator
    - movies[movies.genre.isin(['Action', 'Drama', 'Western'])].head()
    - # Negate the filtering
    - movies[~movies.genre.isin(['Action', 'Drama', 'Western'])].head()
36. Drop columns that has more that 80% missing values
    - # keep the columns that has at least 80% of values not missing
    - ufo.dropna(thresh = len(ufo) * 0.80, axis = 'columns')
37. Expand a series of lists into a Dataframe
    - df = pd.DataFrame({
    -    'col1': ['a', 'b', 'c'],
    -    'col2': [[10, 11], [12, 13], [14, 15]]
    - })
    - df_new = df.col2.apply(pd.Series)
    - pd.concat([df, df_new], axis = 'columns')
38. Return a grouped by series but with the same length of the original dataframe
    - movies.groupby('genre').duration.sum()
    - movies.groupby('genre').duration.transform('sum')
    - movies['total_dur'] = movies.groupby('genre').duration.transform('sum')
    - movies['dur_percent_of_total'] = movies['duration'] / movies['total_dur']
39. Filter the describe output
    - titanic.describe().loc['min':'max', 'Pclass':'Fare']
40. Reset index vs unstack vs pivot table
    - titanic.groupby(['Sex', 'Pclass']).Survived.mean().reset_index()
    - titanic.groupby(['Sex', 'Pclass']).Survived.mean().unstack()
    - titanic.pivot_table(index = 'Sex', columns = 'Pclass', values = 'Survived', aggfunc = 'sum', margins = True)
41. mean of a dataframe
    - df.loc[df.country == 'Italy'].mean(numeric_only = True)
42. reset index will fix the format
    - agg = (df
    - .groupby(["year", "continent"])[["lifeExp", "pop"]]

- .agg(["mean", "std", "count"])
- .reset_index()
-)
- # how to reset the multi_level column hierarchy
- agg.columns = ['_'.join(col).strip() for col in agg.columns.values]

43. Tidy data
    - # link to tidy paper data
    - df = pd.read_csv('Data/pew.csv')
    - # id_vars = the column that we do not want to change in the process
    - df.melt(id_vars = "religion")
    - df_tidy = df.melt(id_vars = "religion", var_name = 'income', value_name = 'count')
    - bil.melt(id_vars = ['year', 'artist', 'track', 'time', 'date.entered'], value_name = 'rank', var_name = 'week')
44. Several methods for splitting and getting
    - df_long[["type", "country"]] = df_long.cd_country.str.split("_", expand = True)
    - # a new feature: get method(0)
    - df_long.cd_country.str.split("_").str.get(1)
45. Convert one columns to separate columns for each category value in the original column
    - df.pivot_table(index = ['id', 'year', 'month', 'day'], columns = 'element', values = 'temp').reset_index()